Indexing Technique for Similarity Matching in Large Video Databases

Sanghyun Park[a], June-Suh Cho[a], and Ki-Ho Hyun[b]

[a] IBM T.J. Watson Research Center, 30 Saw Mill River Rd., Hawthorne, New York, USA
[b] School of Computer & Information Engineering, YoungSan University, Korea

ABSTRACT

Similarity matching in video databases is of growing importance in many new applications such as video clustering and digital video libraries. In order to provide efficient access to relevant data in large databases, there have been many research efforts in video indexing with diverse spatial and temporal features. However, most of the previous works relied on sequential matching methods or memory-based inverted file techniques, thus making them unsuitable for a large volume of video databases. In order to resolve this problem, this paper proposes an effective and scalable indexing technique using a *trie*, originally proposed for string matching, as an index structure. For building an index, we convert each frame into a symbol sequence using a *window order heuristic* and build a disk-resident trie from a set of symbol sequences. For query processing, we perform a depth-first search on the trie and execute a *temporal segmentation*. To verify the superiority of our approach, we perform several experiments with real and synthetic data sets. The results reveal that our approach consistently outperforms the sequential scan method, and the performance gain is maintained even with a large volume of video databases.

Keywords: Indexing technique, Similarity matching, Video databases, Trie, Window order heuristic, Window-based feature

1. INTRODUCTION

In recent years, the growth of the amount of video data in multimedia databases has been tremendous. Similarity matching in video databases is of growing importance in many new applications such as *video clustering* and *digital video libraries*. In order to provide effective and efficient access to relevant data in large databases, there have been many research efforts[2, 4, 6, 8, 10, 12, 16, 18] in indexing of video databases with features such as colors, textures, semantics, and motions.

A large number of previous video indexing approaches depend on the *sequential scan* method to retrieve similar videos in a database. Mohan[11] and Vailaya et. al[17] proposed video sequence matching methods using action similarity and the combination of image content and image motion, respectively. Lienhart et. al[9] and Sanchez et. al[13] used color coherence vectors and principal components of color histograms, respectively, for detecting similar commercial video clips. Adjeroh et. al[1] employed the *vstring* representation for video sequences and introduced *vstring edit distance* as a similarity indicator. Ardizzone et.al[2] extracted a single MPEG motion vector from each 16x16 sub-image, and Meng and Chang[10] used low-level motion features such as zoom and pan of camera. Since all the works mentioned above use the sequential scan method, their performance deteriorates when the database is large.

Squire et. al[14] proposed the use of inverted file techniques for feature-based image retrieval, and Hampapur et. al[6] applied the inverted file techniques to media tracking. Since both approaches keep the inverted files in main-memory during the query processing, they are not suitable for a large volume of video databases.

In this paper, we propose an effective and scalable indexing technique for similarity matching in large video databases using a *trie*,[15] originally proposed for string matching, as an index structure. Assuming that all video frames are divided by the same number of windows, we do the following to build an index: 1) extract a feature from each window and convert it into a discrete symbol, 2) convert each frame into a symbol sequence using a *window order heuristic*, and 3) build a disk-resident trie from a set of symbol sequences. For query processing, we do the following: 1) take the first two steps of index construction to generate a symbol sequence for each target frame, 2) find the data frames closest to each target frame by performing a depth-first search on the trie, and 3) execute a *temporal segmentation* to find the videos or sub-videos similar to a target video.

This paper is organized as follows. Section 2 explains the two features employed in this paper, and Section 3 describes our index construction and query processing methods. Section 4 evaluates the proposed approach through experiments in comparison with the sequential scan method. Finally, Section 5 summarizes and concludes the paper.

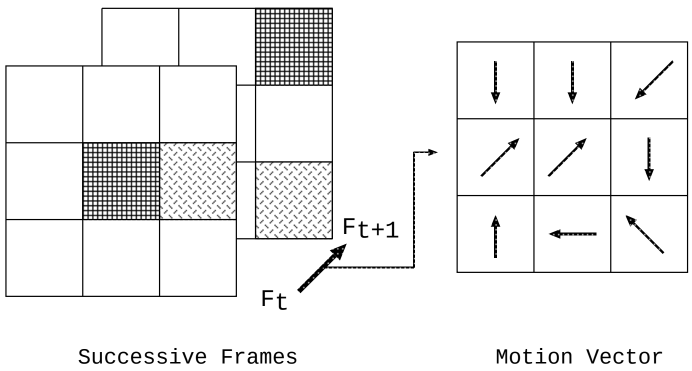
2. FEATURE EXTRACTION

Since we divide every frame into multiple windows, we take window-based feature extraction methods. This section describes the two window-based features, a motion feature and an ordinal feature, employed in this paper.

2.1. Motion Feature

Motion[5,7] is an essential feature of video sequences. By analyzing the motions embedded in a video sequence, we can extract its unique temporal information. The algorithms for estimating the displacement vectors of moving objects can be roughly classified into the three groups such as the optical-flow-based approach, the feature-based approach, and the window-matching-based approach. We employ the window-matching-based approach since it is well suited to our indexing method.

Let F_t and F_{t+1} be the current frame and its subsequent frame, respectively. The frames are partitioned into multiple windows and a motion vector is estimated for each window. To find the displacement in F_{t+1} for a point (x, y) in F_t, the surrounding window of the point is compared to the pixels in the search area. The best match is found according the criterion of the minimum mean-square error. Then, the direction of the motion vector is quantized into q directions or levels. In addition to the q directions, if the motion vector of a window has zero magnitude, the window can also be assigned a level 0. Figure 1 shows the steps of the motion estimation process.



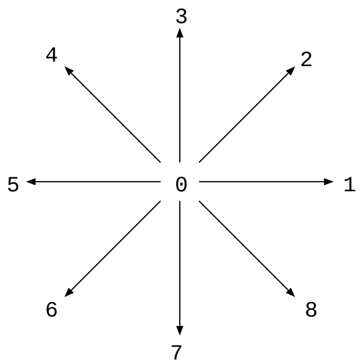


Figure 1: Motion estimation process.

2.2. Ordinal Feature

We employ the ordinal measure[3,11] to capture the spatial information of a video sequence. The ordinal variable is drawn from a discrete ordered set like the grades in schools. The ratio between two measurements is not of consequence; only their relative ordering is relevant. The relative ordering between measurements is expressed by their ranks. A rank permutation is obtained by sorting the sample in ascending order and labeling them using integers $[1, 2, 3.....m]$, m being the size of the sample. The ordinal measure has the advantage that it is less sensitive to a pixel variation. In our application, the average gray level in each window is viewed as an ordinal variable. The set of average intensities is sorted in ascending order and the rank is assigned to each window of a frame. Figure 2 shows an example.

25.3	80.2	89.8
152.3	93.2	125.7
160.3	225.3	230.6

(a) average intensity value of each window in a frame

1	2	3
6	4	5
7	8	9

(b) ordinal features in a frame

Figure 2: Example of ordinal measure.

3. INDEXING METHOD

This section defines the problem we are going to solve, and describes our index construction and query processing methods.

3.1. Problem Definition

Let us first explain the notations and the similarity measures used in this paper. $V = \langle V[1], ..., V[n] \rangle$ denotes a video with n frames and $F = \langle F[1], ..., F[m] \rangle$ denotes a frame with m windows. Given a distance threshold ε, the distance of any two frames F_i and F_j with m windows is defined as the number of window pairs $(F_i[h], F_j[h])$ $(1 \leq h \leq m)$ whose distances are less than or equal to ε. Given a target video V_t and a data (sub-) video V_i with k frames, their distance is defined as the number of frame pairs $(V_t[h], V_i[h'])$ $(1 \leq h, h' \leq k)$ such that $V_i[h']$ has the smallest distance from $V_t[h]$, and 2) they preserve temporal coherence.[6] The proposed similarity measure is exactly same as the one proposed by Hampapur et. al[6] except for a distance tolerance ε. Our similarity measure enables accurate video matching even when videos have different frame rates, and gives end users more flexible control of frame similarity.

Then, the problem we are going to solve is defined as follows: Given a target video V_t, a tolerance ε, and a set of data videos V_i stored in a database, we want to find the sub-videos $V_i[p : q]$ most similar to V_t. Here, $V_i[p : q]$ represents the sub-video of V_i containing frames in positions p through q.

3.2. INDEX CONSTRUCTION

Our indexing approach uses a trie, which was originally proposed for string matching. A *trie* is a tree structure used for indexing a set of keywords. The internal nodes are empty and the edges store symbols. A path from the root to a leaf represents a symbol sequence inserted into a trie. Usually the leaf nodes store the identifiers of symbol sequences. A trie holds all the information contained in the original symbol sequences but common prefixes are stored only once. Thus there is a large potential for compression if symbol sequences contain a large number of common prefixes and the lengths of common prefixes are long. Figure 3(a) shows the trie constructed from two symbol sequences $X = \langle A, B, B, B \rangle$ and $Y = \langle A, A, B, B \rangle$, and Figure 3(b) shows the trie from $Reverse(X) = \langle B, B, B, A \rangle$ and $Reverse(Y) = \langle B, B, A, A \rangle$.

Now let us describe our index construction method. As shown in Figure 4, it consists of the three steps: pre-processing, symbol sequence generation, and trie construction.

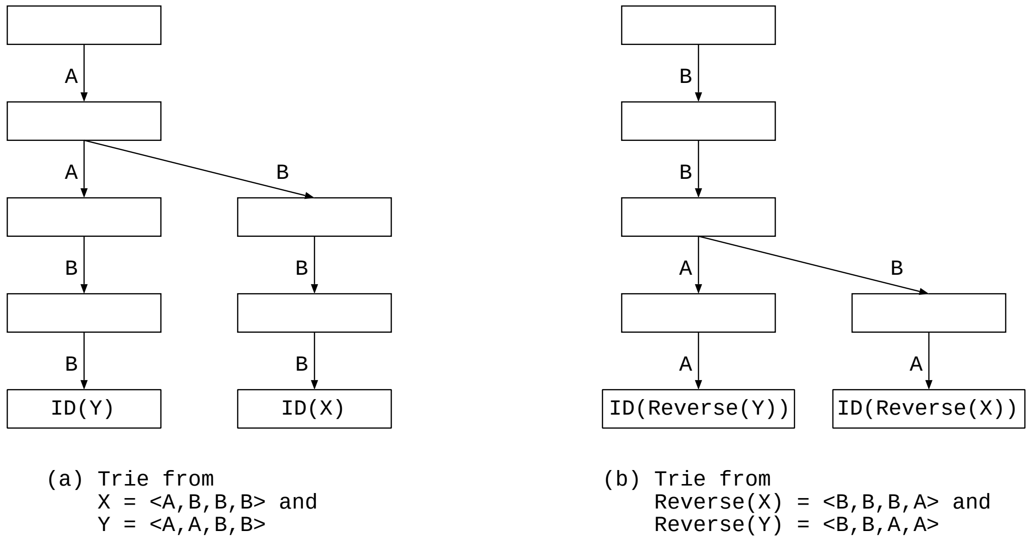


Figure 3: Example tries.

3.2.1. Pre-Processing

The pre-processing steps of an index construction include windowing, feature extraction, and symbolization.

- **Step 1: (windowing)** We divide each frame into a fixed number (for example, 225) of windows.
- **Step 2: (feature extraction)** From each window, we compute a motion feature or an ordinal feature explained in Section 2.
- **Step 3: (symbolization)** We convert each motion or ordinal feature into the corresponding symbol.

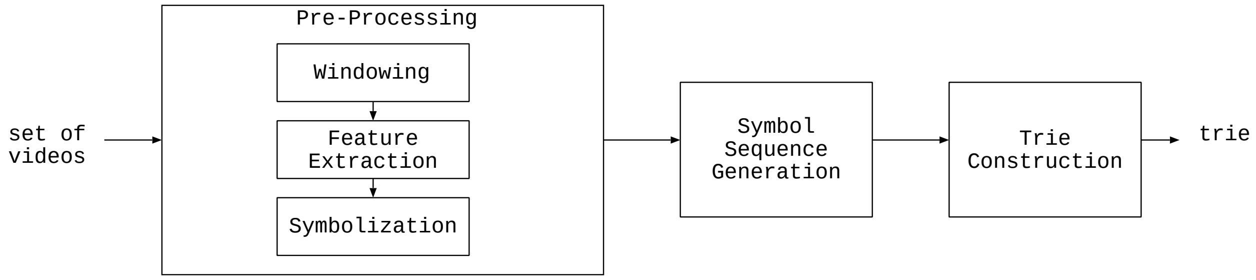


Figure 4: Index construction steps.

3.2.2. Symbol Sequence Generation

To generate a symbol sequence for each frame, this step reads the windows of a frame in the pre-defined order. Notice that the trie becomes more compact when symbol sequences contain more and longer common prefixes. For example, let us compare the two tries in Figure 3. Since the common prefix of $Reverse(X)$ and $Reverse(Y)$ is longer than that of X and Y, the trie from $Reverse(X)$ and $Reverse(Y)$ has fewer nodes.

To maximize the number and the length of common prefixes, we use the following heuristic: 1) for each window, we determine the dominant symbol and compute its occurrence ratio, and 2) we rank the windows according to the occurrence ratios of their dominant symbols. Then the window reading order is determined by this ranking order. We call this heuristic a *window order heuristic*. Figure 5(a) shows 9 windows with their dominant symbols and their occurrence ratios, and Figure 5(b) shows the window reading order determined by our heuristic.

Window-1 A: 80%	Window-2 B: 71%	Window-3 B: 95%
Window-4 A: 64%	Window-5 A: 54%	Window-6 B: 67%
Window-7 C: 85%	Window-8 C: 43%	Window-9 C: 73%

(a) dominant symbols and their occurrence ratios

3	5	1
7	8	6
2	9	4

(b) window reading order

Figure 5: Window reading order with 9 windows.

3.2.3. Trie Construction

This step makes an entry for each frame and insert it into a trie. The index entry for the p-th frame of a video V_i is denoted as (symbol-sequence, i, p). Since every symbol sequence has the same length, we can easily take the disk-based trie construction method: 1) sort index entries according to the ascending order of their symbol sequences, and 2) append each index entry into the trie under construction. This method keeps only the index entry being inserted and the last node of each level inside the main memory, thus making our approach suitable for a large volume of video databases.

3.3. Query Processing

When a target video is submitted with a tolerance ε, we find the most similar sub-videos by taking the four steps shown in Figure 6: pre-processing, symbol sequence generation, index search, and temporal segmentation. During the first two steps, we divide each target frame into the fixed number of windows, compute a feature from each window, and convert the feature into the corresponding symbol.

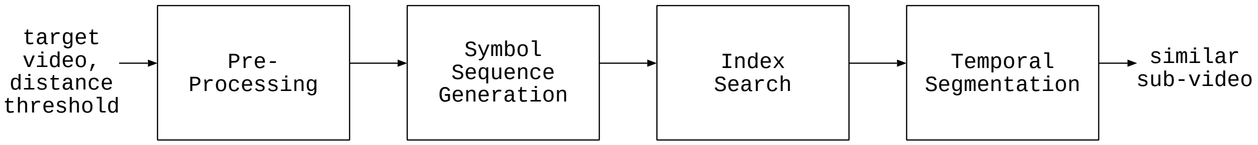


Figure 6: Query processing steps.

3.3.1. Index Search

For each target frame which is now represented as a symbol sequence, we perform a depth-first search on the trie to find the best match which has the largest number of matching symbols with a target frame. Two symbols are matching when their difference is within a distance tolerance ε.

The index search algorithm maintains two global variables, $BestMatch$ and $MinNotMatch$. $BestMatch$ stores the identifiers of the best matching frames found so far and $MinNotMatch$ keeps the number of not-matching symbols of $BestMatch$. Whenever we visit the node N, we inspect the symbol between the node N and its parent node. If that symbol is beyond the distance tolerance ε from the corresponding symbol of the target symbol sequence, we increase the number of non-matching symbols $NotMatch(N)$ of the node N. If

$NotMatch(N)$ is still less than $MinNotMatch$, then we further go down the trie to inspect the children nodes of N. Otherwise, we visit the sibling nodes of N. When we reach the leaf node L whose $NotMatch(L)$ is less than $MinNotMatch$, we replace $BestMatch$ with the identifiers stored in the leaf node L and $MinNumMatch$ with $NotMatch(L)$. Since leaf nodes may store multiple identifiers and different paths may have the same number of non-matching symbols, we may have multiple best matches for each target frame.

3.3.2. Temporal Segmentation

Whenever we find the best match for each target frame, we push it into one or more stacks according to its video and frame numbers. Each stack is labeled with a video number and each element of a stack is labeled with a frame number. For each best match $V_i[p]$, the temporal segmentation step does the following: 1) find the set of stacks labeled with i, and 2) push p onto the set of stacks whose last elements are less than p. After finishing this temporal segmentation, the stack which has the largest number of elements is judged as an answer.

Suppose that a target video V_t has the five frames whose best matches are: $V_1[1]$, $V_2[10]$ for $V_t[1]$, $V_2[5]$ and $V_3[1]$ for $V_t[2]$, $V_2[7]$ and $V_4[7]$ for $V_t[3]$, $V_2[8]$ for $V_t[4]$, and $V_2[9]$ for $V_t[5]$. Figure 7 shows the five stacks after temporal segmentation. Since the second stack is taller than any other stacks, it contains the sub-video most similar to a target video. The starting and ending offsets of the sub-video is obtained by the first and last element values. That is, $V_2[5:9]$ is the answer.

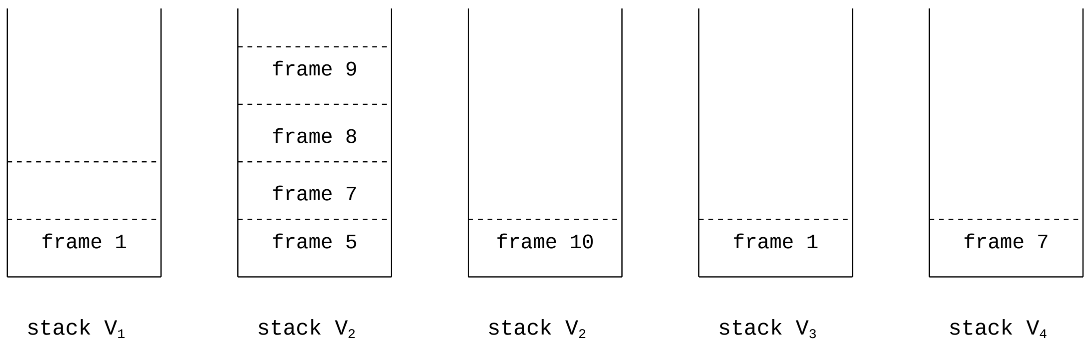


Figure 7: Five stacks for temporal segmentation.

4. PERFORMANCE EVALUATION

This section presents the experimental results for performance evaluation of the proposed approach. Section 4.1 describes the experimental environment and Section 4.2 shows and analyzes the experimental results.

4.1. Experimental Setup

Two kinds of data videos were used for experiments: synthetic and real data videos. Synthetic videos were composed from synthetic frames whose features were taken uniformly from a finite set of integer values. The number of synthetic data videos and their average number of frames were decided according to the purpose of each experiment. We generated 39 real videos from the first 210,600 frames of *Star Wars I* movie, and extracted 540 key frames from each video.

For each experiment, we performed 100 queries with the target videos generated as follows: 1) select one randomly from data videos, 2) extract a sub-video from the selected data video, 3) take a random value from an appropriate range for each window of a frame, and 4) add the value to its corresponding window as a noise. We use an average elapsed time of processing 100 queries as a performance measure.

The hardware platform for the experiments was the IBM Personal Computer 300PL - Pentium III 662 Mhz system equipped with 256 MB RAM, and the software platform was Windows NT 4.0.

	exp 1	exp 2	exp 3	exp 4	exp 5	exp 6
data set	Star Wars I	Star Wars I	Synthetic	Synthetic	Synthetic	Synthetic
Feature	Ordinal	Motion	Uniform	Uniform	Uniform	Uniform
numWins	25	225	9 - 225	16	16	16
numSyms	25	9	8	4 - 32	8	8
distTol	0 - 4	0 - 3	1	1	1	1
numVids	39	39	100	100	100 - 400	100
lenVids	540	540	200	200	200	200 - 1,000
lenQrys	30	30	20	20	20	20

Table 1: Parameter values for each experiment.

For performance evaluation, we compared our approach with the sequential scan method. The sequential scan method reads data videos sequentially to find the best matches for each target frame and then performs the temporal segmentation explained in Section 3.

Table 1 summarizes the parameters used in each experiment where 1) $numWins$ denotes the number of windows in a frame, 2) $numSyms$ is the total number of quantization symbols inside a window, 3) $distTol$ is the distance tolerance given by a user 4) $numVids$ is the total number of videos stored in the database, 5) $lenVids$ is the average number of frames in data videos, and $lenQrys$ is the number of frames in a target video.

4.2. Results and Analysis

Using the ordinal features of Star Wars I, Experiment 1 compares the elapsed times of our approach and the sequential scan with the increasing distance threshold ε from 0 to 4. Experiment 2 does the same thing using the motion feature. As shown in Figure 8 and 9, our approach is about two times faster when using ordinal features and about three times faster when using motion features. Notice that our approach performs better when ε is large. This is because a large ε makes $MinNotMatch$ reach its final value quickly and thus reduces the portion of a trie touched by index search.

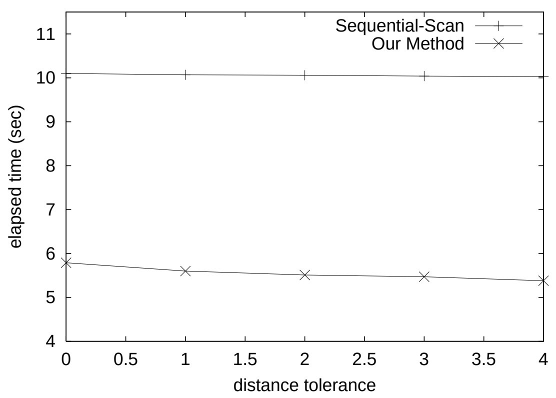


Figure 8. Comparison of elapsed times with increasing distance tolerance. (Star Was I with ordinal features)

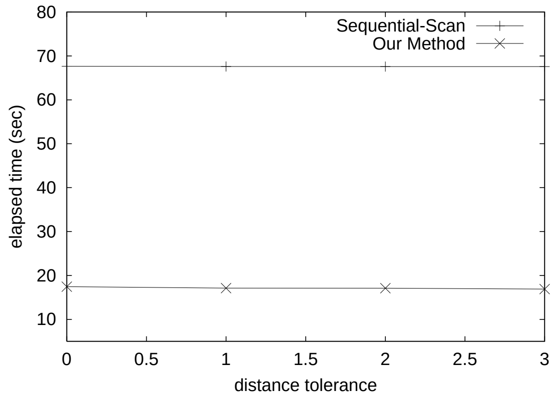


Figure 9. Comparison of the elapsed times with the increasing distance tolerance. (Star Was I with motion features)

Using a synthetic data set, Experiment 3 and Experiment 4 compare the elapsed times of the two approaches with the increasing number of windows from 9 to 225 and with the increasing number of quantization symbols from 4 to 32. As shown in Figure 10, both approaches require more query processing time as the number of windows grows, but the slope of our approach is smaller than that of the sequential scan. As shown in Figure 11, both approaches show the slight increase of query processing time although our approach has sharp increasing pattern when the number of quantization symbols is between 5 and 10. This verifies that the number of common subsequences and their average length reduce as the number of quantization symbols increases. However, the slope becomes steady after the number of symbols exceeds a certain threshold.

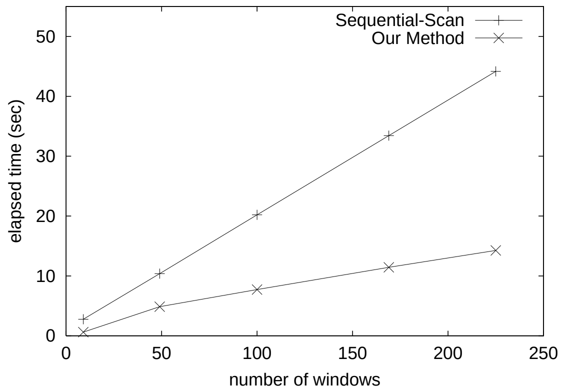


Figure 10. Comparison of elapsed times with increasing number of windows. (synthetic data set)

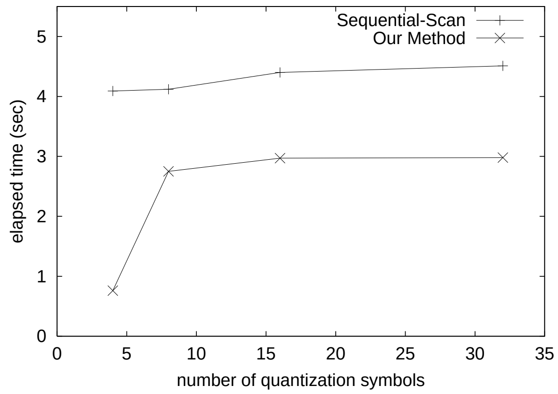


Figure 11. Comparison of the elapsed times with the increasing number of quantization symbols. (synthetic data set)

Using a synthetic data set, Experiment 5 and Experiment 6 compare the elapsed times of the two approaches with the increasing number of videos from 100 to 400 and with the increasing average number of frames in videos from 200 to 1,000. Figure 12 and 13 show the experimental results. Due to the increasing size of video databases, both approaches become slow when the number of videos and their average number frames increase. However, the slope of our approach is smaller than that of the sequential scan. Therefore, our approach is more scalable and better suited for large video databases.

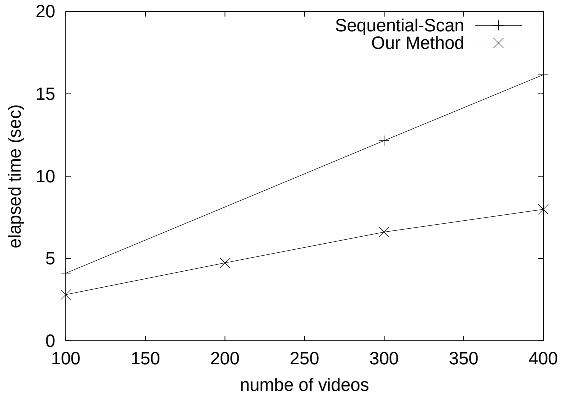


Figure 12. Comparison of the elapsed times with the increasing number of videos. (synthetic data set)

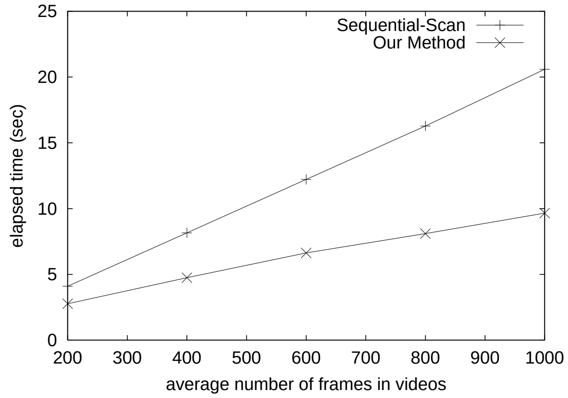


Figure 13. Comparison of the elapsed times with the increasing average number of frames in videos. (synthetic data set)

5. CONCLUSION

Similarity matching in video databases is of growing importance in many new applications. Although there have been many research efforts for providing efficient access to relevant data in video databases, most of the previous works relied on sequential matching methods or memory-based inverted file techniques, thus making them unsuitable for a large volume of video databases.

In order to resolve this problem, this paper proposed an efficient and scalable indexing technique using a trie as an index structure. Our contributions are: 1) adapt a trie for video indexing, and 2) introduce a window order heuristic for reducing the index size and thus improving the search performance. To verify effectiveness and scalability of our approach, we performed the experiments with real and synthetic video databases. The experimental results reveal that our approach consistently outperforms the sequential scan method, and the performance gain is maintained even with a large volume of video data.

REFERENCES

1. D. A. Adjeroh, M. C. Lee, and I. King, "A Distance Measure for Video Sequence Similarity Matching", *Proc. Int'l Workshop on Multi-Media Database Management Systems*, 1998.
2. E. Ardizzone, M. La Cascia, A. Avanzato, and A. Bruna, "Video Indexing Using MPEG Motion Compensation Vectors", *Proc. IEEE Int. Conf. on Multimedia Computing System*, 1999.
3. D. N. Bhat and S. K. Nayar, "Ordinal Measures for Image Correspondence", *IEEE Trans. on PAMI*, 20(4), pp. 415-423, 1998.
4. S. Dagtas and A. Ghafoor, "Indexing and Retrieval of Video based on Spatial Relation Sequences", *Proc. of ACM Multimedia 1999*, 1999.
5. A. Hampapur, K. Hyun, and R. Bolle, "Comparison of Sequence Matching Techniques for Video Copy Detection", *Proc. SPIE Conf. on Storage and Retrieval for Media Databases*, 2002. (accepted to appear)
6. A. Hampapur and R. Bolle, "Feature Based Indexing for Media Tracking", *Proc. Int'l Conf. on Multimedia and Expo*, 2000.
7. M. Ioka and M. Kurokawa, "A Method for Retrieving Sequences of Images on the basis of Motion Analysis", *Proc. SPIE Conf. on Image Storage and Retrieval Systems*, 1994.
8. V. Kobla, D.S. Doermann, K-I. Lin, and C. Faloutsos, "Compressed domain video indexing techniques using DCT and motion vector information in MPEG video", *Proc. SPIE Conf. on Storage and Retrieval for Image and Video Databases*, 1997.
9. R. Lienhart, C. Kuhmunch, and W. Effelsberg, "On the Detection and Recognition of Television Commercials", *Proc. IEEE Conf. on Multimedia Computing and Systems*, 1997.
10. J. Meng and S.-F Chang, "CVEPS - A Compressed Video Editing and Parsing System", *Proc. of ACM Multimedia 1996*, 1996.
11. R. Mohan, "Video Sequence Matching", *Proc. Int'l Conf. on Audio, Speech and Signal Processing*, 1998.
12. Emile Sahouria and Avideh Zakhor, "Motion Indexing of Video", *Proc. International Conference on Image Processing*, 1997.
13. J. M. Sanchez, X. Binefa, J. Vitria, and P. Radeva, "Local Color Analysis for Scene Break Detection Applied to TV Commercials Recognition", *Proc. Visual 99*, 1999.
14. D. M. Squire, H. Muller, and W. Muller, "Improving Response Time by Search Pruning in Content Based Image Retrieval System, Using Inverted File Techniques", *Proc. IEEE Workshop on Content Based Image and Video Libraries*, 1990.
15. G. A. Stephen, *String Searching Algorithms*, World Scientific Publishing, 1994.
16. Roland Tusch, Harald Kosch, and Laszlo Boszormeny, "VIDEX: An Integrated Generic Video Indexing Approach", *Proc. of ACM Multimedia 2000*, 2000.
17. A. Vailaya, W. Xiong, and A. K. Jain, "Query by Video Clip", *Proc. Int'l Conf. on Pattern Recognition*, 1998.
18. J. Wei, Z.N. Li, and I. Gertner, "A novel motion-based active video indexing method", *Proc. IEEE Int. Conf. on Multimedia Computing System*, 1999.